20 VIDEOLEZIONI DI BASIC
PER IMPARARE COL VIC 20

GRUPPO
EDITORIALE
JACKSON

19

COMMODORE
VIC 20

VIDEO BASIC VIC 20
Pubblicazione quattordicinale
edita dal Gruppo Editoriale Jackson
Direttore Responsabile:
Giampietro Zanga
Direttore e Coordinatore Editoriale: Roberto Pancaldi
Autore: Softidea - Via Indipendenza 88 - Como
Redazione software:
Francesco Franceschini, Enrico Braglia, Fabio Calanca
Segretaria di Redazione:
Marta Menegardo
Progetto grafico:
Studio Nuovaidea - Via Longhi 16 - Milano
Impaginazione:
Silvana Corbelli
Illustrazioni:
Cinzia Ferrari, Silvano Scolari
Fotografie:
Marcello Longhini
Distribuzione: SODIP
Via Zuretti, 12 - Milano
Fotocomposizione: Lineacomp S.r.l.
Via Rosellini, 12 - Milano
Stampa: Grafika '78
Via Trieste, 20 - Pioltello (MI)
Direzione e Redazione:
Via Rosellini, 12 - 20124 Milano
Tel. 02/6880951/5

Autorizzazione alla pubblicazione Tribunale di Milano n° 422 del 22-9-1984
Spedizione in abbonamento postale Gruppo II/70 (autorizzazione della Direzione Provinciale delle PPTT di Milano).
Prezzo del fascicolo L. 8.000
Abbonamento comprensivo di 5 raccoglitori L. 165.000
I versamenti vanno indirizzati a: Gruppo Editoriale Jackson S.r.l. - Via Rosellini, 12 20124 Milano, mediante emissione di assegno bancario o cartolina vaglia oppure utilizzando il c.c.p. n° 11666203.
I numeri arretrati possono essere richiesti direttamente all'editore inviando L. 10.000 cdu. mediante assegno bancario o vaglia postale o francobolli.
Non vengono effettuate spedizioni contrassegno.

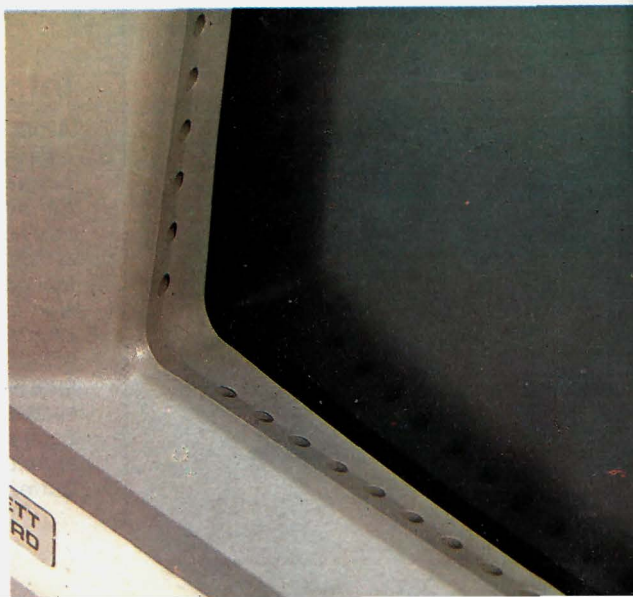
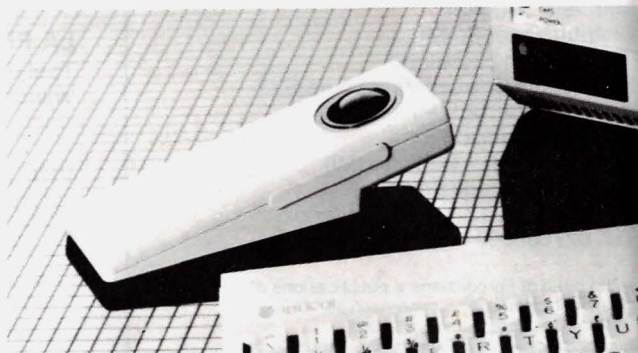
Gruppo Editoriale Jackson

SOMMARIO

Introduzione

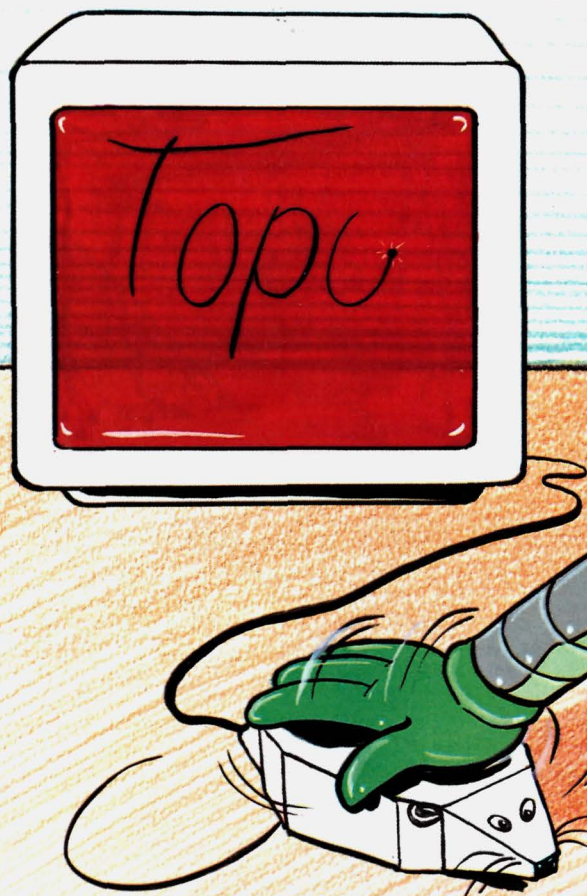
Avete ormai a disposizione tutti gli elementi per essere dei buoni programmatori in BASIC.
Per ottenere dal computer, però, risultati ancora più strabilianti la strada è una sola: il linguaggio macchina.
Non tutto per fortuna deve essere programmato autonomamente; molte volte, anzi, è preferibile in termini di fattibilità, tempi e costi, acquistare software standard.
Ecco allora word processor, apreadsheet, data base e altri "attrezzi" di grande utilità.

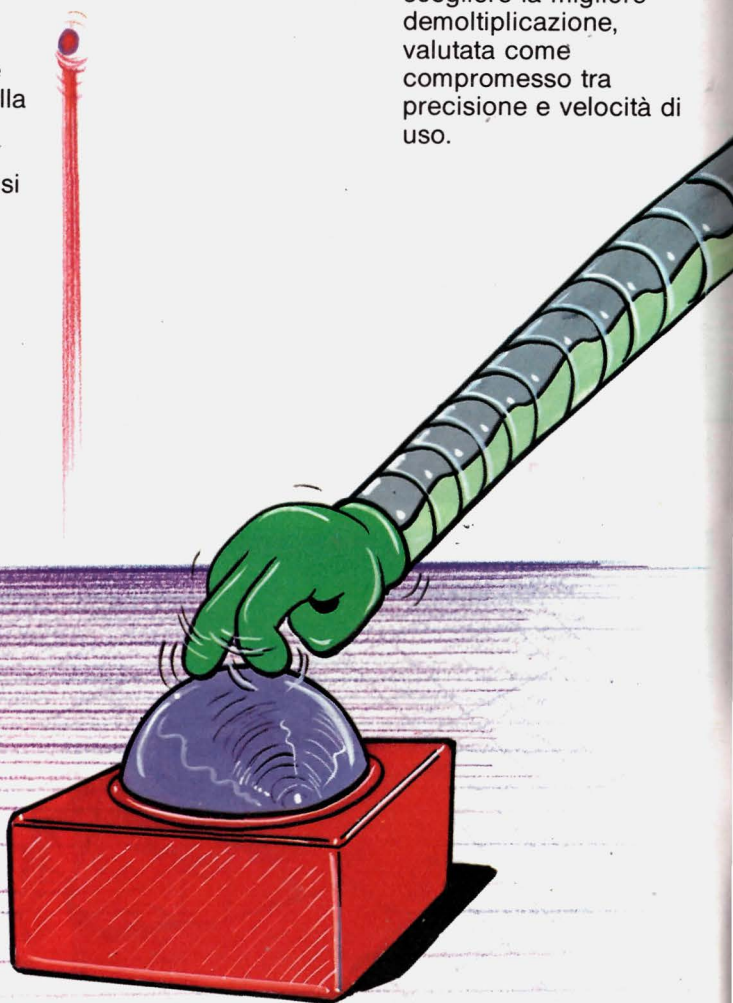
Moderni dispositivi di input

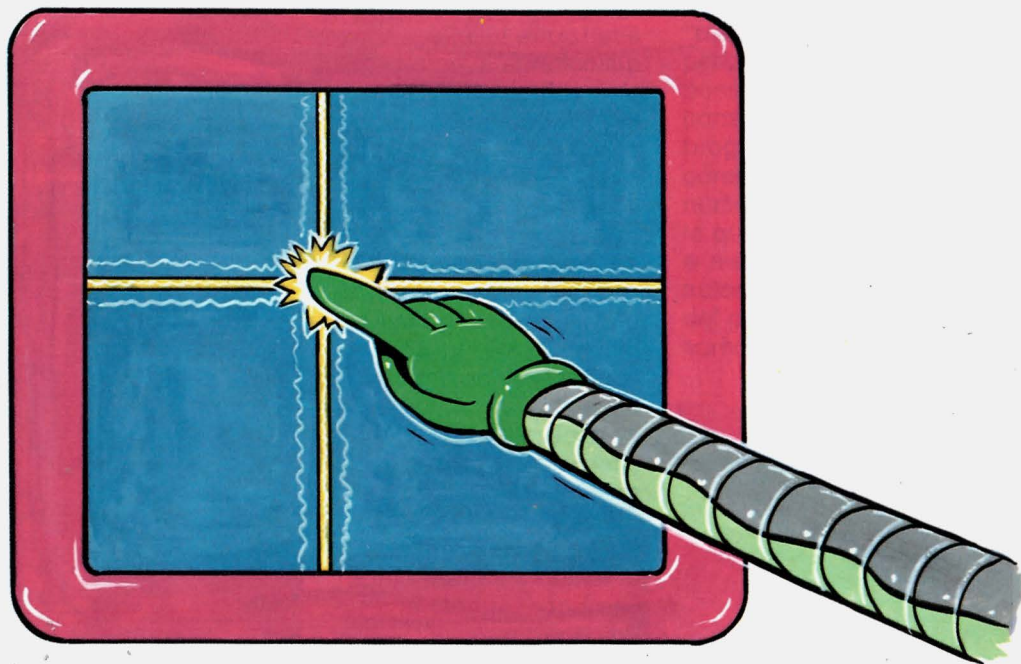
Come abbiamo già avuto modo di vedere, per qualsiasi computer la tastiera costituisce un dispositivo di input estremamente importante, anzi fondamentale.
Esistono tuttavia numerose altre periferiche di input che consentono, di fornire al computer in modo semplice e immediato, i dati che si desidera immettere nella memoria.
Da quando la grafica è entrata prepotentemente nel campo dei personal computer, si sono sviluppati nuovi dispositivi per rendere più semplice ed immediato il colloquio uomo-macchina.
I più diffusi sono il mouse, la trackball e il touch-screen.

MOUSE

Il mouse ("topolino") è della trackball capovolta. La sfera viene infatti posta in rotazione spostando l'intero dispositivo su un piano liscio (per esempio una scrivania).

Rispetto alla trackball, il mouse richiede uno spazio maggiore, essendo necessaria una certa libertà di movimento per la manovra: in compenso l'operazione risulta più intuitiva, in quanto lo spostamento del cursore sullo schermo ricalca fedelmente quello imposto con la mano. È inoltre possibile muovere il cursore tenendo contemporaneamente premuto il pulsante (o i pulsanti) del mouse, cosa impossibile con la trackball.

Per i suoi vantaggi di precisione e facilità di impiego, il mouse si sta attualmente affermando come la principale periferica di input della nuova generazione di personal computer.

TRACKBALL

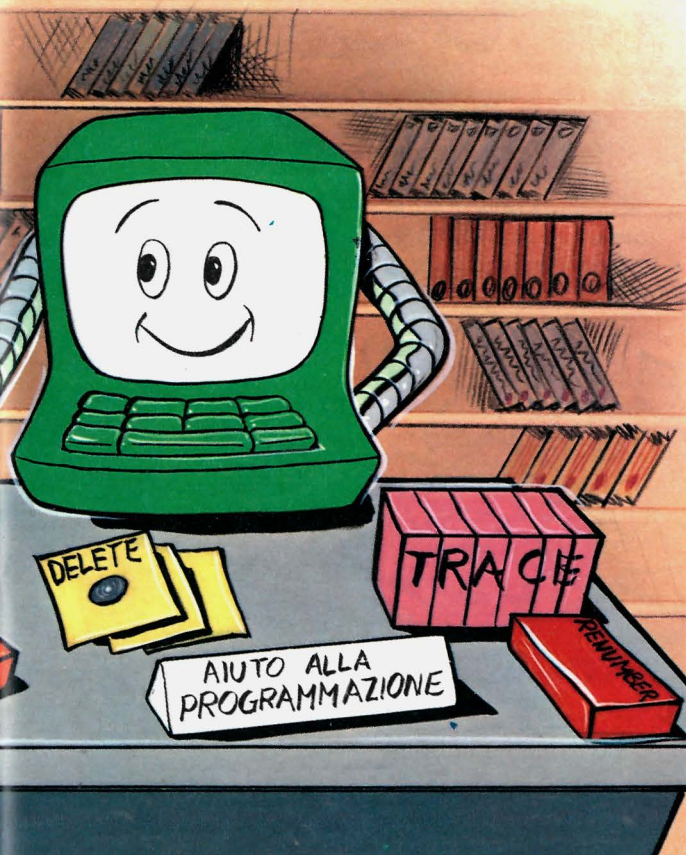
La trackball (letteralmente "sfera tracciante") è una periferica costituita da una sfera liscia, completamente libera di ruotare in un supporto fisso; viene posta in rotazione utilizzando il palmo della mano. Due rullini posti a fianco della sfera (all'interno del supporto) rilevano la rotazione lungo due assi ortogonali e la convertono in una serie di impulsi elettrici, in numero proporzionale all'angolo di rotazione. Dato che - a differenza di joystick e paddle - non esiste fine corsa, è possibile ottenere tutta la precisione che si desidera, anche se talvolta questo vantaggio va a scapito della velocità di utilizzo.
È pertanto compito del software di "pilotaggio" scegliere la migliore demoltiplicazione, valutata come compromesso tra precisione e velocità di uso.

TOUCH-SCREEN

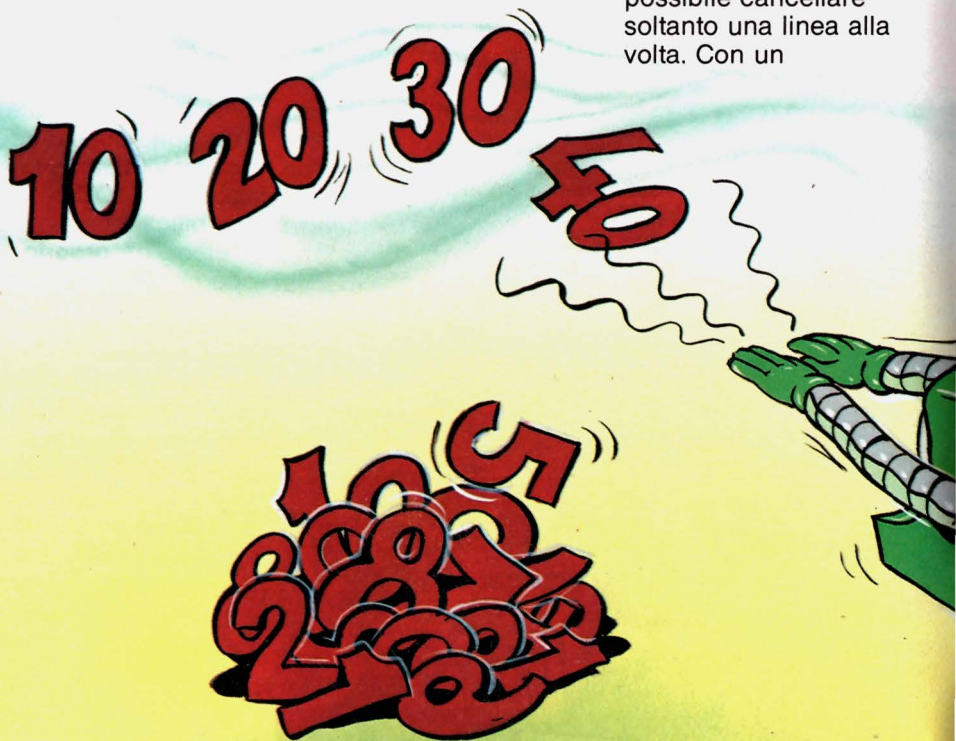
L'idea del touch-screen è molto simile a quella della penna luminosa: touch-screen significa infatti "video sensibile al tatto".
In realtà non è lo schermo video ad essere sensibile al tocco dell'operatore, ma un foglio trasparente che ricopre lo schermo stesso. Un reticolo di sensori ottici rende sensibile questo foglio al tocco delle dita o di una penna.
Il maggior svantaggio di questa soluzione è costituito dalla scarsissima risoluzione del dispositivo. Inoltre, e questo costituisce uno scoglio notevole, lo strato di grasso sempre presente sulla pelle finisce con l'accumularsi sul video, pregiudicando la lettura dello schermo.
Di contro, l'uso di questa periferica risulta estremamente naturale ed intuitivo, potendo quindi essere utilizzata anche da persone non troppo esperte.

Software di ausilio

Qualunque computer, per quanto sofisticato e perfezionato, da solo non è in grado di operare: occorre infatti abbinare alle caratteristiche hardware della macchina il software appropriato alla risoluzione del problema (o dei problemi) che si intende risolvere.
Per perseguire questo obiettivo, tuttavia, il programmatore deve compiere un insieme di passi ed operazioni variabili di volta in volta.
Occorre quindi analizzare e valutare sotto ogni minimo aspetto il problema, esaminando con attenzione tutte le difficoltà da superare.
Una volta individuato nei termini generali il problema, le varie funzioni da svolgere vengono man mano evidenziate in appositi schemi (comunemente definiti - ormai lo sai benissimo - diagrammi a blocchi), che individuano la successione logica delle azioni, con le possibili situazioni alternative e gli eventuali momenti di decisione. In questa fase si decide anche se l'intero insieme di funzioni da realizzare verrà concentrato in un unico programma oppure suddiviso in più programmi e relativi sottoprogrammi.
Una volta stabiliti i diagrammi a blocchi il programma passa infine alla vera e propria fase di programmazione, realizzata mediante la scrittura delle varie istruzioni in un linguaggio che - come il BASIC - possa essere "capito" dalla macchina.

LINGUAGGIO

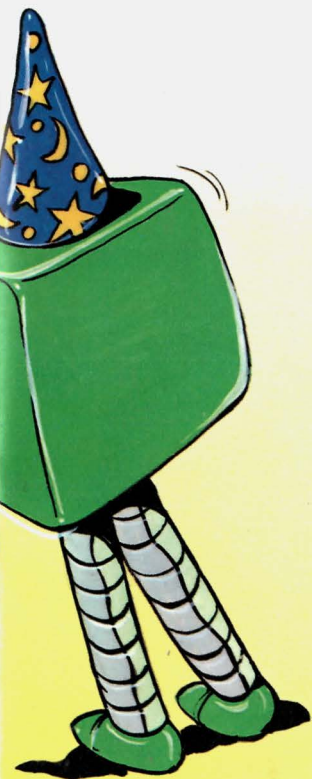
Terminata la programmazione si provvede quindi (dopo aver preventivamente registrato il programma su un sicuro supporto magnetico a avviare la successiva fase di collaudo.
A questo punto inizia il momento forse più stimolante e impegnativo dell'intero procedimento. È in questa fase che tutti i nodi vengono al pettine, evidenziando le eventuali lacune che una analisi frettolosa del problema può essersi lasciata alle spalle.
In tale operazione possono sovente essere di aiuto al programmatore quegli insiemi di programmi, comunemente definiti come "software di ausilio", che permettono di ispezionare, valutare e correggere gli errori in modo molto più veloce ed agevole.
I programmi di ausilio più comuni riguardano e permettono operazioni a prima vista molto banali: al momento opportuno essi diventano comunque pressoché indispensabili, specialmente quando si desidera sfruttare in pieno le possibilità del proprio calcolatore.
Ne esistono molti in commercio; anche sulle riviste specializzate essi vengono spesso presi in considerazione. I più completi TOOL-KIT (tool significa letteralmente "utensile", proprio perché questi programmi possono essere considerati gli arnesi del programmatore) consentono di svolgere numerose operazioni, tra le quali:

- **numerare** automaticamente le linee del programma mentre si scrive. Si evita così di

dover inserire tutte le volte i numeri di linea; naturalmente, è possibile specificare il passo - cioè l'incremento - che si desidera avere tra una linea e l'altra;

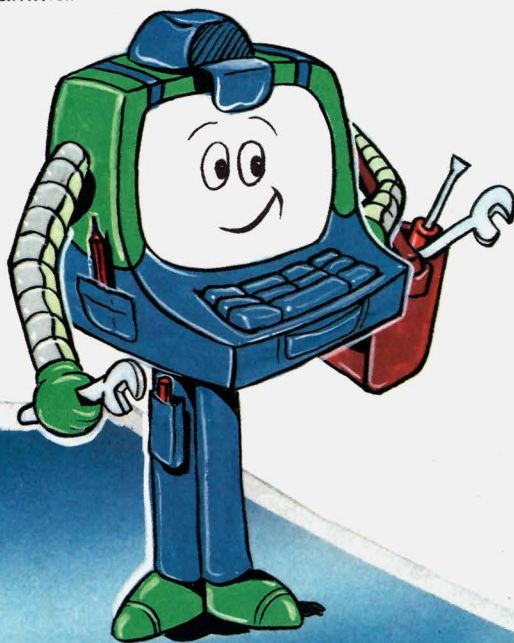
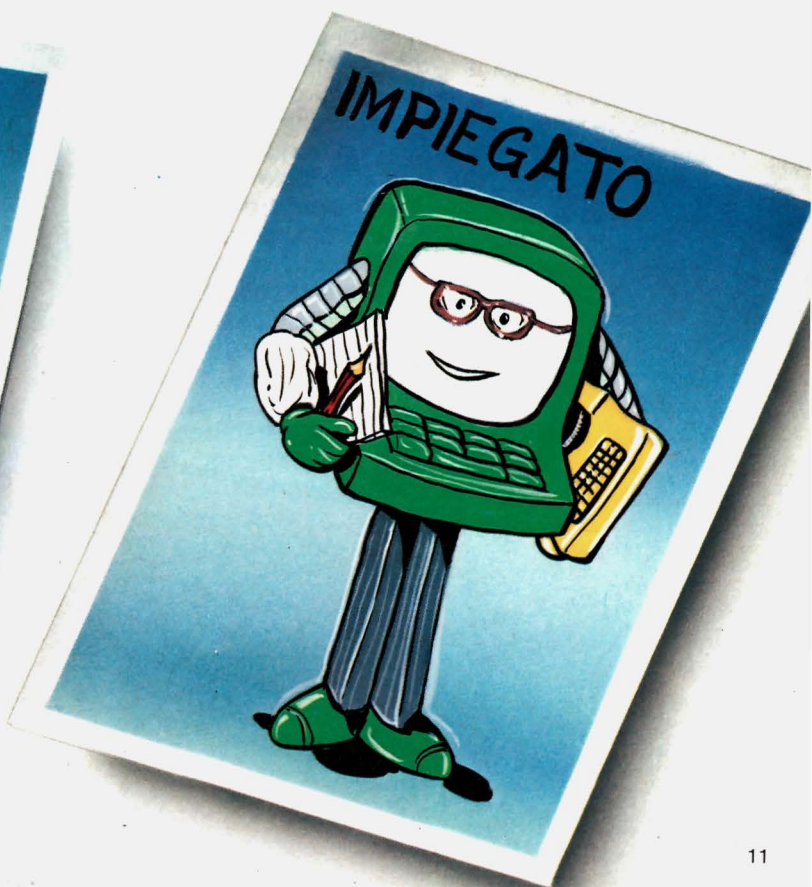
- **rinumerare** un programma che abbia subìto molti aggiustamenti. Questa è una delle possibilità più apprezzate dai programmatori, che spesso - se non esistessero questi programmi - non avrebbero più modo di inserire nuove istruzioni. È chiaro che la rinumerazione delle linee deve implicare - per lo meno in un programma professionale - anche la corretta rinumerazione in corrispondenza dei vari comandi GOTO e GOSUB;
- **cancellare** blocchi di programma. Molti elaboratori permettono di effettuare questa azione: da sistema operativo nel tuo VIC 20 è invece possibile cancellare soltanto una linea alla volta. Con un

programma di "delete" (cancellazione) è possibile dare comandi del tipo "delete 10-130" (cancella tutte le linee comprese tra la 10 e la 130);

- **ricercare** una particolare stringa di caratteri nel programma. Si evita in questo modo di dover faticosamente ricercare la stringa (per esempio un nome di variabile) per tutta la lunghezza del listato;
- **sostituire** automaticamente una stringa di caratteri con un'altra;
- **trovare** errori di sintassi. Molti programmi raggiungono infatti tali livelli di complessità logica da provocare in casi molto speciali l'esecuzione di certe istruzioni. Eliminando subito (cioè prima dell'esecuzione) tutti gli errori di sintassi, si può risparmiare il tempo che normalmente si rende necessario per arrivare in quelle istruzioni, correggerle e ricominciare di nuovo;
- **visualizzare** i numeri delle linee di programma via via che esse vengono eseguite.

Anche questa possibilità è di estremo interesse e utilità: accade infatti molto spesso che il programma giri correttamente, cioè che fornisca dei risultati in uscita, ma che questi non siano assolutamente corrispondenti a quanto dovrebbe realmente accadere.
Sono questi gli errori più subdoli e difficili da individuare, in quanto non riguardano la sintassi del programma, bensì la sua logica. Con un programma di "trace" (traccia) è allora possibile seguire - attraverso la visualizzazione progressiva dei numeri di linea che vengono eseguiti - lo svolgimento del programma in tempo reale, cioè nel momento stesso in cui esso sta funzionando. Molte volte è sufficiente un'occhiata a questi numeri per essere in grado di risolvere un guaio provocato da un errato salto di riga o da una mancata successione di istruzioni;

- **fondere programmi** diversi in un unico programma. L'utilità in questo caso è indiscutibile: a chi non è capitato di voler

utilizzare una stessa subroutine in due diversi programmi? Il programma di "merge" (letteralmente: funzione) è forse - insieme a quello di renumber - il più utilizzato ed apprezzato. Esso provvede infatti ad "accodare" al programma che si trova in memoria in un certo momento un secondo programma (normalmente letto da un supporto magnetico), evitando sovrapposizione di istruzioni o perdita di righe di programma.

Supporti commerciali

Proprio come accade per i capi di abbigliamento, i programmi possono essere su misura o preconfezionati. Questi ultimi, ovviamente, costano meno.
I programmi specificamente progettati e scritti per la soluzione di un determinato problema sono quelli che svolgono il loro compito con la massima velocità, con il migliore sfruttamento delle caratteristiche tecniche del computer utilizzato e con la maggiore rispondenza alle esigenze di chi li deve

adoperare.
I vantaggi di questi programmi possono essere riassunti in pochi ma importanti punti:
— sono realizzati professionalmente e collaudati;
— sono pronti immediatamente (non richiedono cioè lunghi tempi di attesa);
— hanno un prezzo abbastanza contenuto.
Analizzando anche il rovescio della medaglia, lo svantaggio principale di questi programmi è che essi non possono soddisfare al cento per cento le esigenze di chi li usa. Esiste infatti sempre un certo margine di "insoddisfazione", dovuto al fatto che un programma progettato per risolvere un problema nel suo complesso non può tener conto di tutti i casi particolari.
Un programma di contabilità - per esempio - non potrà essere identico per un artigiano o per una media industria: per questo molti programmi sono

allora "aperti", cioè permettono un margine più o meno ampio di modificabilità, per eventuali "personalizzazioni".
Ci sono invece esigenze che sono realmente comuni ad un numero molto vasto di utenti, come la scrittura e l'archiviazione dei dati.
Infatti i più importanti "pacchetti applicativi" disponibili attualmente sul mercato sono: i word processor, i fogli elettronici e i data base.

Word processor

Un programma per l'elaborazione dei testi (dall'inglese word processor) è una delle possibili applicazioni che, da sola, può addirittura giustificare l'acquisto di un personal computer.
La caratteristica principale di un word processor è infatti quella di trasformare l'elaboratore in una macchina da scrivere molto speciale. Permette infatti di scrivere di getto, quasi alla rinfusa, tutto ciò che si desidera, con la possibilità di modificare qualsiasi parte del testo senza dover riscrivere tutto da capo al momento di battere la bella copia e consentendo di ottenere con la stessa qualità di stampa un numero illimitato di copie. Il suo uso abolisce quindi in maniera definitiva gomme, correttori, carta carbone e cose simili.
Un sistema di trattamento della parola gestisce inoltre automaticamente l'allineamento delle parole sia all'interno delle righe che dei margini, i quali possono essere variati a piacimento con pochi ed elementari comandi. Ma la caratteristica

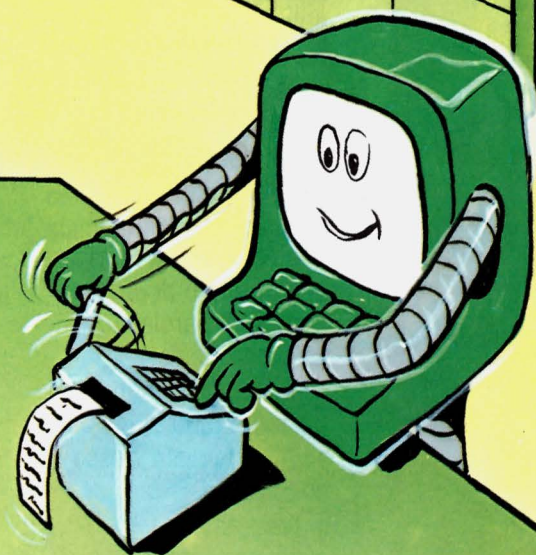
fondamentale di un programma word processor è comunque quella di permettere la correzione dinamica di tutto ciò che si è scritto: si può cioè "tornare indietro" e correggere direttamente sul video, lasciando al computer e alla sua stampante il compito di riscrivere l'intero testo su carta.
Le funzioni più importanti e necessarie di un sistema per il trattamento di testi sono:

- a capo automatico. Non è quindi necessario controllare il fine riga, poiché le parole eccedenti la capienza della stessa vengono automaticamente trasferite all'inizio della riga successsiva;
- possibilità di inserimento di caratteri, parole o righe nel testo già esistente, per correzioni o aggiunte;
- cancellazione di caratteri, parole e frasi, con eliminazione automatica dello spazio resosi disponibile nella riga;
- possibilità di reimpaginare automaticamente il testo, sia perché si è voluto variare il numero di caratteri per riga sia perché si è variata la spaziatura tra le righe;
- possibilità della ricerca automatica di una parola o di un gruppo di parole nell'intero testo;
- possibilità di copiare e trasferire parti di testo in altra parte dello stesso documento.

In programmi maggiormente evoluti esistono altre funzioni, più particolari di quelle appena accennate, ma non per questo di minore utilità: il principio generale di funzionamento resta comunque sempre lo stesso. Naturalmente, qualsiasi word processor permette di salvare il testo su un supporto magnetico.

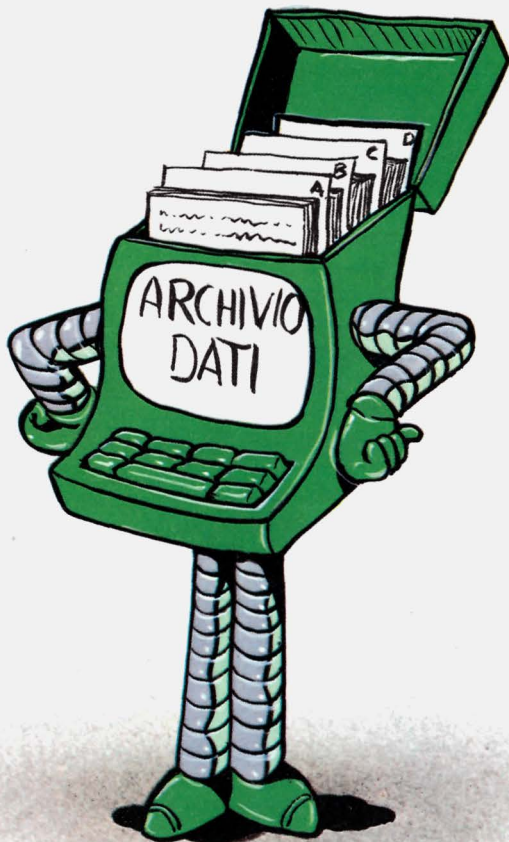
Fogli elettronici

Una delle applicazioni che ha avuto maggior fortuna e diffusione nel settore degli home e personal computer è sicuramente il trattamento automatico dei prospetti di tipo tabellare (quelli, per intenderci, organizzati per righe e colonne), cioè il cosiddetto foglio di calcolo elettronico o worksheet (talvolta spreadsheet).
L'utilizzo pratico di un tale sistema è molto ampio e vario: va dalle

GENN.
FEBB.
MARZO
APR.
MAGGIO
GIU.
AFFITTO
CASA
GAS
BENZINA
LUCE
TOTALE

simulazioni di vendita, in funzione dei livelli di produzione, alla gestione del bilancio personale o familiare, oppure all'esame dell'andamento dei costi, ricavi e utili di un'attività commerciale nel corso dei dodici mesi dell'anno.
Ma vediamo brevemente come funziona.
Si tratta di un insieme di *righe e colonne* individuate da un codice (ad esempio, lettere per le colonne e numeri per le righe, proprio come nel gioco della battaglia navale). Ciascuna coordinata può contenere un dato numerico, una formula che collega ed elabora algebricamente o logicamente più dati, o un insieme di lettere qualsiasi, quali quelle componenti un titolo. Tra le diverse informazioni *che possono comparire* in due o più caselle possono essere definite *delle relazioni analitiche*, algebriche o di altro tipo (per esempio statistico). Ciò fatto, è sufficiente introdurre tramite tastiera i vari valori nelle caselle definite come variabili principali, perché tutte le variabili dipendenti vengano automaticamente calcolate dal programma e fatte comparire nelle caselle pertinenti.
Da questa caratteristica deriva la capacità dei pacchetti di spreadsheet di fungere da potenti strumenti di "simulazione" per modelli di tipo statistico o finanziario. L'acquisto di un foglio di lavoro elettronico è pertanto consigliabile in tutti i casi in cui esista una esigenza, ragionevolmente frequente, di elaborare prospetti tabellari con dati eventualmente tra *loro correlati.*

Data base

I data base (o, con una libera traduzione italiana, i programmi per l'organizzazione di dati) sono programmi di utilizzo generalizzato, adatti per reperire e memorizzare dati e notizie, archiviare e classificare pratiche di ufficio o insiemi di informazioni e così via.
Un data base permette di gestire *elettronicamente* qualsiasi tipo di archivio di informazioni, disposto e *strutturato secondo il* desiderio di chi utilizza il programma. Un esempio renderà l'idea nel migliore dei modi.
Supponiamo di voler comporre una rubrica telefonica elettronica, contenente - oltre al nome e al cognome (ed ovviamente al numero telefonico) delle varie

persone - anche l'indirizzo e la città di residenza.
Con il data base dopo aver introdotto gli elementi è possibile ordinare la lista delle persone, per esempio in ordine alfabetico oppure per zone di residenza, per prefisso telefonico o per qualsiasi altra "chiave".
È inoltre immediatamente ricercabile, in modo assolutamente semplice e automatico, "la persona che ha prefisso X e abitazione con numero civico Y".
Per tornare a un discorso generale, le applicazioni di un data base sono praticamente infinite: tante quanti possono essere gli utilizzi di archivi di informazione. La cosa importante di questi programmi è che essi permettono la gestione di qualsiasi tipo di informazione, non importa quanto strana o complicata, ricercandola ed aggiornandola con la massima rapidità e automazione.

Tecniche di indirizzamento

Il 6502 non dispone di una grossa varietà di istruzioni (complessivamente sono 56). Sotto questo aspetto numerose altre CPU risultano nettamente superiori, arrivando facilmente ad avere un numero di istruzioni eseguibili quasi doppio. Ciò nonostante, è uno dei microprocessori più diffusi. La ragione di questo fatto è molto semplice: tutto è dovuto all'ampia possibilità di tecniche di indirizzamento disponibili, che incrementa il numero effettivo delle istruzioni a 152.
Vediamo innanzitutto cosa intendiamo con l'espressione "tecniche di indirizzamento". Abbiamo già detto che l'Assembler è un linguaggio a basso livello, scarsamente (anzi, per nulla) strutturato e dotato di forme logiche veramente minime. In altre parole, l'Assembler si avvale di istruzioni con estrema semplicità operativa ed esecutiva: tutto viene difatti basato sui numeri e sulle locazioni di memoria.
Proprio in virtù di questo fatto, per poter lavorare al meglio in qualsiasi parte della memoria, i costruttori di microprocessori cercano di introdurre la massima flessibilità nelle operazioni eseguibili, fornendo alla CPU le più vaste tecniche di indirizzamento.
L'indirizzamento fa pertanto riferimento al modo in cui deve essere considerato l'operando in una certa istruzione, influenzando di conseguenza anche il risultato della operazione stessa. Esamineremo adesso, uno per uno, i diversi indirizzamenti disponibili, valutandone le possibilità, gli utilizzi, i vantaggi e gli svantaggi.

Indirizzamento implicito

L'indirizzamento implicito è utilizzato da un'istruzione a singolo byte, che opera sui registri interni del microprocessore. Un esempio di istruzione implicita è DEX, che significa: "decrementa di 1 il contenuto del registro X". Come puoi notare, il modo di indirizzamento implicito non richiede operando; quest'ultimo è infatti già "implicitamente" contenuto nell'operatore.

È un indirizzamento che assicura un'elevata velocità di esecuzione: ogni volta che le istruzioni implicite operano esclusivamente sui registri della CPU richiedono infatti soltanto due cicli di clock per essere eseguite. Quando accedono a qualche locazione della memoria vengono invece richiesti tre cicli. (nota: per ciclo di clock si intende il tempo necessario alla CPU per svolgere una operazione elementare. Tale tempo viene scandito inflessibilmente da un oscillatore (una specie di orologio al quarzo in miniatura). Tieni presente che - a titolo di informazione - il 6502 esegue più di un milione di operazioni elementari ogni secondo: e questo per tutto il tempo che tieni acceso il computer!

Indirizzamento immediato

Questa forma di indirizzamento è utilizzata per caricare con uno specifico valore l'accumulatore o i registri indice (convenzionalmente chiamati X e Y). Tutte le istruzioni nel modo a indirizzamento immediato sono perciò lunghe due byte. Il primo contiene il codice operativo ed il secondo la costante numerica da caricare nel registro (o da utilizzare con uno dei registri) per una operazione aritmetica o logica.
Per esempio, se noi volessimo caricare nell'accumulatore il valore 160 (esadecimale A0) in modo immediato, potremmo scrivere:

```
LDA #$A0
```

Questa istruzione dice: "carica nell'accumulatore (LoaD Accumulator) il valore esadecimale A0". Il simbolo # serve per ricordarci la nostra necessità che questa operazione venga eseguita in modo immediato.

Indirizzamento assoluto

L'indirizzamento assoluto è il modo in cui i dati sono normalmente recuperati dalla memoria. È specificato dal codice operativo, seguito da un indirizzo a 16 bit. L'indirizzamento assoluto richiede pertanto istruzioni di tre byte. Un esempio di questa forma di indirizzamento è:

```
STA $1234
```

Questa istruzione specifica che il contenuto dell'accumulatore deve essere memorizzato (STore Accumulator) alla locazione di memoria numero 1234 esadecimale.
Lo svantaggio dell'indirizzamento assoluto è di richiedere un'istruzione di tre byte, cioè è un modo abbastanza lento. Per migliorarne l'efficienza viene allora reso disponibile un altro modo di indirizzamento, quello diretto, nel quale per l'indirizzo viene però utilizzata una sola parola.

Indirizzamento diretto

In questo modo il codice operativo è seguito da un indirizzo a 8 bit. Il vantaggio di questo approccio è di richiedere solo due byte invece dei tre dell'indirizzamento assoluto. Lo svantaggio è la limitazione dovuta al fatto che tutti gli indirizzi, in questo modo, devono essere compresi tra 0 e 255.

Poiché convenzionalmente è possibile dividere le locazioni di memoria in gruppi di 256 (per esempio 65535 locazioni - pari a 64 K - possono essere considerate come 256 "pagine" da 256 byte l'una), questo modo di indirizzamento viene anche chiamato "indirizzamento di pagina zero". Esso fa infatti riferimento alla prima pagina (chiamata pagina zero), cioè alle prime 256 locazioni di memoria. Un esempio di questo modo è

```
LDA $FB
```

che significa: "carica l'accumulatore con il valore contenuto all'indirizzo numero 00FB esadecimale".

Indirizzamento relativo

Questo è un modo di indirizzamento spesso utilizzato per le istruzioni di salto (cioè per i comandi che trasferiscono l'esecuzione da un punto all'altro del programma). Per definizione l'indirizzamento relativo utilizza due byte: il primo è un'istruzione di salto, mentre il secondo specifica lo spostamento e il suo segno. Cosa significa? Supponiamo per esempio di voler mandare l'esecuzione da un certo punto del programma a un altro. Per fare questo dobbiamo specificare (oltre naturalmente all'istruzione di salto) anche il numero di locazioni di cui vogliamo spostarci. Poiché lo spostamento può essere sia positivo che negativo (cioè può avvenire sia in avanti che indietro), l'istruzione di salto - che al massimo deve interessare 255 locazioni - può essere al limite negativa di 128 (salto all'indietro di 128 locazioni), oppure positiva di 127 (salto in avanti di 127 locazioni). Naturalmente, il più grosso svantaggio di questo modo è che non permette salti superiori alle 128 locazioni. Tuttavia, poiché la maggior parte dei cicli tende ad essere breve, la diramazione può essere utilizzata quasi sempre e aiuta a risolvere il problema.

Indirizzamento indicizzato

L'indirizzamento indicizzato è una tecnica particolarmente pratica per accedere, uno dopo l'altro, ai dati contenuti in un gruppo di locazioni. Il principio è che l'istruzione risulta specificata sia da un indirizzo che da un registro indice. Il contenuto dell'indice viene sommato all'indirizzo specificato per ottenere l'indirizzo finale. Questo viene fatto perché sono disponibili istruzioni specializzate per incrementare o decrementare separatamente ciascuno dei registri indice.
Eseguendo un ciclo di incremento dell'indice, è quindi possibile accedere con poco lavoro ad ampie zone di memoria.
Ecco un esempio della tecnica dell'indirizzamento indicizzato:

```
LDA $1500, Y
```

L'istruzione dice al microprocessore di sommare il valore del registro indice Y con il valore $1500 (per ottenere l'indirizzo finale) e quindi di memorizzare nell'accumulatore il valore contenuto in questo indirizzo.

Indirizzamento indiretto

Esistono numerosi casi in cui non è necessario conoscere un certo indirizzo della memoria *fino a quando il* programma viene fatto eseguire. In questo caso è utile riferirsi a un certo byte della memoria, prelevando da esso non il valore desiderato, bensì l'indirizzo a cui trovare questo valore.
Questo caso è analogo alla situazione in cui diverse persone devono entrare in una casa ed esiste solo una chiave.
Per convenzione la chiave della casa sarà nascosta sotto il vaso.
Ogni utilizzatore deve guardare sotto il vaso per trovare la chiave della casa (ovvero per trovare l'indirizzo desiderato).
Gli esempi che vedremo tra breve illustreranno qualche possibile applicazione di queste tecniche.

Incremento/ decremento

La più semplice espressione aritmetica eseguibile dal 6502 è l'operazione algebrica di addizione e sottrazione del valore 1 dal valore contenuto in un registro. A prima vista questa possibilità non sembra aprire orizzonti sconfinati: in realtà la disponibilità di una simile operazione permette la costruzione di strutture ben più complesse, come per esempio i cicli FOR...NEXT del BASIC.
L'importanza dei comandi di addizione e sottrazione è talmente grande che esistono addirittura due istruzioni specifiche per portare a termine questi compiti.
Di ciascuna di queste esistono tre diverse forme:

INC incrementa di 1 il valore posto in uno specifico indirizzo di memoria
INX incrementa di 1 il valore nel registro X
INY incrementa di 1 il valore nel registro Y
DEC decrementa di 1 il valore posto in uno specifico indirizzo di memoria
DEX decrementa di 1 il valore nel registro X
DEY decrementa di 1 il valore nel registro Y.

Tutte queste istruzioni possono influenzare alcuni bit del registro di *stato, precisamente il bit* di segno negativo e il bit zero.

Il bit di segno viene settato se il bit più significativo del registro (cioè quello della posizione 7) viene posto a 1 a seguito di un incremento o un decremento altrimenti vale zero.
Il FLAG di zero viene invece settato soltanto se, a seguito di una operazione, il registro chiamato in causa contiene il valore zero.
Per esempio, incrementando di 1 il valore $FF (255 esadecimale), si porterà a $00 il valore del registro, verrà posto a 1 il FLAG di zero (Z=1) e si annullerà il FLAG di segno negativo. D'altro canto, decrementando un registro contenente $00, si porterà il valore del registro a $FF, ponendo contemporaneamente a 1 il FLAG di segno e a zero il FLAG di zero.

I cicli

I cicli sono una parte importantissima - anzi, vitale - in qualsiasi *linguaggio di* programmazione: in Assembler lo sono ancora di più poiché molte istruzioni, che in linguaggio ad alto livello richiedono un unico comando, per la loro esecuzione in codice macchina necessitano invece di sequenze e ripetizioni più o meno lunghe.
Prima di passare alla dimostrazione pratica dell'uso dei cicli in Assembler occorre però introdurre altri due gruppi di istruzioni: quelle di confronto e quelle di salto.

I confronti

I registri X, Y e A (A sta per Accumulatore) possono essere confrontati con il contenuto di una locazione della memoria (oppure con un valore numerico qualsiasi) mediante le tre istruzioni

CMP confronta l'accumulatore
CPX confronta il registro X
CPY confronta il registro Y

Il risultato del confronto altera i FLAG di zero (Z), di segno (N) e di riporto, o carry, (C) del registro di stato. In che modo? *La risposta è in questa* breve tabella:

REGISTRO	CARRY	ZERO	SEGNO
minore del dato	0	0	1
uguale al dato	1	1	0
maggiore del dato	1	0	0

Confrontando per esempio il valore dell'accumulatore (supponiamo che sia $5A) con il numero $B3, dalla tabella ricaviamo che il valore dei vari FLAG sarà:

C=0 Z=0 N=1

Sulla base di questi risultati potremo quindi prendere le decisioni più appropriate.

I salti

Una volta effettuata l'operazione di confronto può essere *necessario* eseguire un'operazione di salto (così come in BASIC scriviamo per esempio IF B>A THEN GOTO 300). Le istruzioni che in Assembler permettono i salti condizionati (cioè attuabili in seguito a determinate condizioni) sono:

— BMI (salta se meno) e BPL (salta se più). Queste istruzioni si basano sul valore del flag di zero.
— BCC (salta se il riporto è zero) e BCS (salta se il riporto è 1). Esse controllano il contenuto del flag di riporto - o di carry - per effettuare il salto.
— BEQ (salta quando il risultato è zero) e BNE (salta quando il risultato non è zero). In questo caso si basano sul valore del flag di zero.

— BVS (salta quando l'overflow è posto a 1) e BUC (salta quando l'overflow è zero). Stavolta le istruzioni verificano il contenuto di V, o bit di overflow. Il flag di overflow è un altro bit del registro di stato che viene posto a 0 oppure a 1 in corrispondenza di determinate condizioni.

Queste istruzioni operano la decisione e il salto all'interno della stessa istruzione. Tutti i salti specificano uno spostamento relativo, cioè in avanti o indietro di +127 locazioni o di −128 byte. Per esempio

```
BNE – 14
```

significa: "se il risultato di confronto (che sarà già stato operato in precedenza) ha posto il flag Z a 1, allora esegui un salto all'indietro di 14 byte".
Oltre a quelle appena viste, sono disponibili anche altre due istruzioni di salto: JMP e JSR.
JMP fa eseguire il salto a una nuova locazione (che viene specificata mediante un indirizzo a 16 bit), JSR è invece la chiamata a una subroutine. Il loro funzionamento è sostanzialmente identico ai comandi BASIC GOTO e GOSUB; la differenza importante rispetto alle istruzioni di salto viste in precedenza è comunque che JMP e JSR sono comandi di salto incondizionato, cioè vengono eseguiti in qualsiasi caso.
Naturalmente, così come il comando GOSUB necessita di RETURN, anche l'istruzione JSR richiede che al termine del sottoprogramma sia possibile ritornare al programma principale.
Per questo esiste l'istruzione RTS, che abbiamo già utilizzato quando - al termine di un programma Assembler - volevamo ritornare al BASIC.
Lavorando in Assembler, RTS fa ritornare al punto esattamente successivo a quello che ha provocato la chiamata alla subroutine: JSR e RTS devono quindi lavorare sempre in coppia.
Terminata finalmente la parte teorica passiamo adesso a quella pratica, cioè all'applicazione dei concetti visti finora.
Cominciamo con un esempio sull'indirizzamento assoluto.
Proviamo, anziché ricorrere alla solita istruzione del BASIC, PRINT, a scrivere due lettere sullo schermo (per esempio due "A") mediante il linguaggio macchina. Per fare questo dobbiamo caricare il codice corrispondente al carattere prescelto nella memoria di schermo e nella memoria colore. Il

programma Assembler sarà allora:

```
LDA #$01     ;carica nell'accumulatore il display code di "A"
STA $1E50    ;memorizza A nelle due locazioni
STA $1E51    ;della memoria di schermo
STA $9650    ;memorizza il colore bianco
STA $9651    ;nella memoria colore
RTS          ;ritorna al BASIC
```

Carichiamo il linguaggio macchina utilizzando il solito programma BASIC:

```
10 POKE 643,128:POKE 644,29
15 LET INIZIO=7552
20 LET FINE=8000
30 RESTORE : PRINT CHR$(147)
40 FOR A=INIZIO TO FINE
50 READ X:IF X=999 THEN GOTO 80
60 POKE A,X
70 NEXT A
80 SYS INIZIO
90 DATA...qui vanno scritti i codici...
```

Per inserire i codici del *programma in linguaggio* macchina nella linea DATA dobbiamo eseguire la conversione delle istruzioni mnemoniche Assembler nei corrispondenti valori numerici:

ASSEMBLER	CODICE ESADECIMALE	CODICE DECIMALE
LDA #$01	A9,01	169,1
STA $1E50	8D,50,1E	141,80,30
STA $1E51	8D,51,1E	141,81,30
STA $9650	8D,50,96	141,80,150
STA $9651	8D,51,96	141,81,150
RTS	60	96

ricordando che il nostro *programma* BASIC, per non essere interrotto da un messaggio di errore, richiede che l'ultimo valore sia il numero 999. Eseguendo il programma, vedrai comparire sullo schermo le due lettere nella posizione corrispondente alle due locazioni che saranno state interessate dalla routine in linguaggio macchina.

L'esempio che abbiamo appena visto è abbastanza interessante, ma non molto pratico: se infatti volessimo far apparire sullo schermo non più soltanto due, bensì duecento caratteri, il programma assumerebbe lunghezze inaccettabili. Occorre pertanto ricorrere a un ciclo, che automatizzi il processo e lo renda molto più breve. Così come in BASIC scriveremmo (senza ricorrere a un ciclo FOR):

```
10 I=0
20 PRINT "&";
30 I = I+1
30 IF I < > 200 THEN GOTO 20
40 END
```

in Assembler potremo allora fare:

```
LDX #$00
LDA #$26
JSR $FFD2
INX
CPX #$C8
BNE $F8
RTS
```

Cerchiamo adesso di capire il significato delle varie istruzioni: poiché useremo come contatore *del ciclo il registro* X, *lo* inizializziamo a zero con l'istruzione LDX #$00.

Carichiamo quindi *nell'accumulatore il* valore 38 (26 esadecimale), corrispondente al codice del carattere "&".
A questo punto iniziamo il ciclo vero e proprio: eseguiamo innanzitutto una chiamata alla subroutine posta a partire dalla locazione 65490 (FFD2 esadecimale). Tale subroutine - che fa parte del sistema operativo ed è *quindi scritta in* memoria ROM - è la stessa che utilizza anche l'interprete BASIC quando esegue un'istruzione PRINT; il suo compito è quello di visualizzare sullo schermo il carattere il cui codice è in quel momento contenuto *nell'accumulatore.* Ad ogni chiamata essa stamperà quindi un carattere "&", visto che è a quest'ultimo che corrisponde il valore posto nell'accumulatore. Il ciclo prosegue incrementando il registro indice (INX) e confrontando il numero ottenuto con 200 (CPX#$C8): se il risultato del confronto non è positivo (cioè se il *registro* X *non è ancora* stato incrementato 200 volte), l'esecuzione

ritorna indietro, alla chiamata della subroutine. L'istruzione

```
BNE $F8
```

dice infatti al microprocessore: "se il risultato del confronto che hai appena eseguito non è ancora verificato, allora esegui un salto all'indietro di 8 byte". Il codice esadecimale $F è il valore - *fornito dal costruttore* della CPU - che abbinato all'istruzione di salto provoca il decremento di 8 byte del program-counter, cioè del registro adibito al controllo dello svolgimento del programma.
L'esecuzione del ciclo prosegue quindi per 200 volte, finché il confronto non diventa verificato: a questo punto l'intero compito sarà stato *eseguito e l'istruzione* RTS restituirà il controllo all'interprete BASIC. La conversione delle varie istruzioni nei corrispondenti valori decimali porta a questi numeri (da inserire nella linea DATA):

```
162, 0, 169, 38, 32, 210, 255, 232, 224, 200, 208, 248, 96.
```

Se invece avessimo voluto scrivere un ciclo senza fine analogo a questo:

```
10 PRINT "&";
20 GOTO 10
```

avremmo potuto scrivere:

```
LDX #$ 00
LDA #$ 26
JSR $FFD2
CPX #$ C8
BNE $F7
RTS
```

Poiché - rispetto al programma precedente - abbiamo eliminato *l'istruzione di incremento* del registro X, il risultato del confronto eseguito da CPX#$C8 non sarà mai verificato e il programma continuerà all'infinito, senza mai poter raggiungere il fatidico RTS. In questo caso come sola possibilità per terminare *l'esecuzione resterà* unicamente quella di spegnere il computer, con la conseguenza di perdere anche il programma appena scritto. I codici numerici di questa routine sono:

```
162, 0, 169, 38, 32, 210, 255, 224, 200, 208, 247, 90.
```

Tool di programmazione

Visto che ormai stiamo addentrandoci sempre di più all'interno del VIC, possiamo cominciare ad esaminare alcuni "programmi di utilità", brevi routine e suggerimenti, che possono sempre essere di aiuto in particolari circostanze.
Innanzitutto qualche "trucchetto" che può sempre diventare prezioso.
1) Per poter posizionare il cursore in una qualsiasi posizione dello schermo basta inserire nei programmi questa breve routine:

```
POKE 781, R:POKE 782,C:POKE 783,0:SYS 65520
```

Assegnando alle variabili R e C, rispettivamente, il valore della riga (compresa fra 0 e 21) e della colonna (tra 0 e 22) a cui desideri inviare il cursore, con queste 4 istruzioni avrai la possibilità di spostarti ovunque sullo schermo.
2) Per impedire il listato di un programma è sufficiente eseguire una semplice istruzione POKE:

```
POKE 4097,0 (VIC in versione base)
POKE 1025,0 (VIC con espansione 3K)
POKE 4609,0 (VIC con espansione ≧ 8K)
```

3) Per impedire il funzionamento del tasto STOP bisogna fare una

```
POKE 788,PEEK (788)+3
```

(questo comando funziona solo se non è già stato eseguito in precedenza).
Per ripristinare il normale funzionamento è invece necessario comandare una

```
POKE 788, PEEK (788)–3
```

4) Per conoscere dove il VIC ha memorizzato una certa variabile (cioè per saperne l'indirizzo) occorre eseguire questo breve programma BASIC (supponendo che A sia la variabile "incognita")

```
10 P=A
20 A=PEEK (71)+256*
   PEEK(72)–2
30 PRINT A
40 A=P
```

5) Per aumentare leggermente la larghezza dello schermo (25 colonne ogni riga) esegui:

```
POKE 36864,9:POKE 36866,153:POKE 36867,40:
POKE 36865,42
```

Sopprimendo i margini con queste POKE, otterrai le colonne aggiuntive.
Adesso vediamo invece un programma di utilità: esso consente di listare i programmi una riga alla volta, permettendoti di muoverti avanti e indietro nel listato. Nel caso tu volessi correggere qualche errore, ciò non ti sarà però possibile: dovrai annotare la riga da modificare e correggerla dopo aver terminato il programma.
Come puoi notare, la numerazione di questa routine parte da 60000. Per farla funzionare devi infatti "accodarla" al programma che desideri listare. La numerazione così alta difficilmente potrà disturbare le linee del tuo programma.

```
60000 A=PEEK(44)*256+PEEK(43)-1:B=A
60002 C=PEEK(A+3)+PEEK(A+4)*256
60003 PRINT"♥ E GOTO 60010" : PRINT "LIST"; C
60004 POKE 631,19:POKE 632,17:POKE 633,144:POKE 634,13:
      POKE 635,19:POKE 636,13:POKE 198,6
60005 END
60010 IF PEEK(197)=5 THEN 60100
60020 IF PEEK(197)=61 THEN 60200
60030 GOTO 60010
60100 D=(PEEK(A+1)+PEEK(A+2)*256)-1
60110 IF (PEEK(D+1)+PEEK(D+2)*256) <> 0 THEN A=D
60120 GOTO 60002
60200 IF A=B THEN 60002
60220 A=A-1: IF PEEK(A)=0 AND PEEK(A-4) <> 0 AND
      PEEK(A-3) <> 0 THEN 60002
60230 GOTO 60200
```

Per lanciare il programma esegui un RUN 60000 oppure un GOTO 60000; i tasti con

i quali potrai lavorare sono il "+" e il "–".
Il carattere "" in linea 60003 si ottiene premendo contemporaneamente SHIFT e CLR/HOME; "" con CTRL e 2.

Esaminiamo adesso un altro programma, di genere completamente diverso da quello appena visto, ma non per questo meno utile: è infatti una routine di MERGE, che ti permetterà di unire in un unico blocco di istruzioni due distinti programmi.
Ecco il listato:

```
10 POKE 55,176:POKE 56,29
20 A=29*256+177:B=7601
30 FOR I=B TO B+78:READ C:POKE I,C:NEXT:NEW
40 DATA 169,0,133,10,32,209,225,165
50 DATA 43,72,165,44,72,56,165,45
60 DATA 233,2,133,43,165,46,233,0
70 DATA 133,44,169,0,133,185,166,43
80 DATA 164,44,169,0,32,213,255,176
90 DATA 14,134,45,132,46,32,51,197
100 DATA 104,133,44,104,133,43,96,170
110 DATA 201,4,144,244,240,10,104,133
120 DATA 44,104,133,43,24,108,0,3
130 DATA 164,186,136,240,209,208,239
```

La lunga serie di DATA corrisponde naturalmente a una routine in linguaggio macchina, protetta da eventuali sconfinamenti del BASIC mediante l'abbassamento (linea 10) dei puntatori che indicano la fine della memoria.
La versione appena scritta è valida per i VIC in versione base: per i VIC con espansioni di memoria 8K RAM e occorre modificare in questo modo le linee 10 e 20:

```
10 POKE 55,176:
   POKE 56,29
20 A=63*256+172:
   S=16300
```

Per configurazioni diverse dalla base e dalla 8K i puntatori devono essere modificati di conseguenza.
Per utilizzare la routine batti innanzitutto il listato e salvalo su nastro o su disco. Quindi, quando hai intenzioni di effettuare un MERGE, carica ed esegui il programma.
A questo punto ti sarà possibile fondere nuove istruzioni al termine di qualsiasi programma che già si trovi in memoria. La sintassi da

rispettare per eseguire l'unione dei due blocchi è molto simile a quella del comando LOAD.

```
SYS 7601 "nome programma", numero dispositivo
```

Nella versione 8K RAM il valore 7601 va sostituito con 16300.
Se per caso impartissi il comando SYS senza avere ancora nessun programma in memoria, il risultato sarà analogo a un LOAD.

Velocità del L/M

Se ancora ce ne fosse bisogno, questo programma dimostra la rapidità di esecuzione del linguaggio macchina. Confrontala con quella di un analogo programma BASIC che stampi agli stessi 255 caratteri.

```
10 DATA 162, 0, 169, 81, 157, 0, 30
15 DATA 169, 2, 157, 0, 150, 232, 208, 243, 96
20 FOR C = 0 TO 15
25 READ A : POKE 828 + C, A
30 NEXT C
35 PRINT "□" : SYS 828
40 END
```

VIDEOESERCIZI

Divertiti a scoprire l'effetto ed il possibile utilizzo di queste semplici routine.

```
10 REM PROVA DI UNA ROUTINE IN ROM
20 FOR I = 1 TO 19 : PRINT I : NEXT
30 R = INT (RND (0) * 22)
40 PRINT CHR (10) CHR (17) TAB (R) "*";
50 FOR I = 1 TO L
60 SYS 59765
70 NEXT
```

```
10 PRINT CHR$ (19) CHR$ (17) CHR$ (157) CHR$ (148)
20 POKE 218, 158
30 R = INT (RND (0) * 22)
40 PRINT CHR$ (19) CHR$ (17) TAB (R) "*";
50 GOTO 10
```